Fermi National Accelerator Laboratory

FERMILAB-Conf-91/337

Software Development for a Switch-based Data Acquisition System

A. Booth*, D. Black and D. Walsh

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

**SSC Laboratory*
2550 Beckleymeade Ave., Dallas, Texas

December 1991

* Presented at the *IEEE Nuclear Science Symposium*, Santa Fe, New Mexico, November 2-9, 1991.

Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CHO3000 with the United States Department of Energy

Disclaimer

Software Development for a Switch-based Data Acquisition System

Alexander Booth*, Dennis Black & Don Walsh
Fermi National Accelerator Laboratory**

Abstract

We report on the software aspects of the development of a switch-based data acquisition system at Fermilab. This paper describes how, with the goal of providing an "integrated systems engineering" environment, several powerful software tools were put in place to facilitate extensive exploration of all aspects of the design. These tools include a simulation package, graphics package and an Expert System shell which have been integrated to provide an environment which encourages the close interaction of hardware and software engineers. This paper includes a description of the simulation, user interface, embedded software, remote procedure calls, and diagnostic software which together have enabled us to provide real-time control and monitoring of a working prototype switch-based data acquisition (DAQ) system. The prototype DAQ system is capable of running at 600Hz of current CDF events (200Kbytes), and comprises several VME Sun 1E boards, as well as the switch backplane and auxiliary components.

1.0 Introduction

With the ever-increasing complexity of detectors and their associated data acquisition systems, it is important to bring together a set of tools to enable system designers, both hardware and software, to understand the behavioral aspects of the system as a whole, as well as the interaction between different functional units within the system. For complex systems, human intuition is inadequate since there are simply too many variables for system designers to begin to predict how varying any subset of them affects the total system. On the other hand, exact analysis, even to the extent of investing in disposable hardware prototypes, is much too time consuming and costly. Simulation bridges the gap between physical intuition and exact analysis by providing a learning vehicle in which the effects of varying many parameters can be analyzed and understood. Simulation techniques have been used in the development of the Scalable Parallel Open Architecture Data Acquisition System at Fermilab. This paper describes the work undertaken at Fermilab in which several sophisticated tools have been brought together to provide an integrated systems engineering environment specifically aimed at designing DAQ systems.

At Fermilab, the Scalable Parallel Open Architecture Data Acquisition System [1] project was directed at addressing the needs of future High Energy Physics (HEP) Experiments which require much greater bandwidth than those of previous era's. A protoype data acquisition system (figure 1) has been implemented. It consists of eight test transmitters (which emulate the front end electronics) and transmit their data to eight Input Time Slot Interchangers (TSI), which buffer the data before feeding it into the switch. Eight Output TSI's receive data in parallel from the switch and send complete events onto a farm of processors (emulated by the output TSI's in the prototype system).

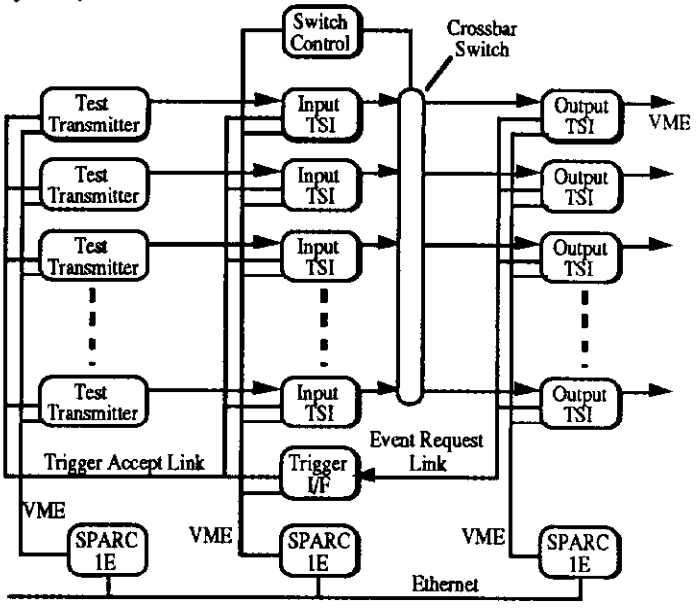


Figure 1 Prototype Switch System

From the outset of the project a goal was to provide an integrated systems engineering environment in which hardware and software development could proceed in parallel and actually complement one another. To achieve this, it was necessary to bring together a set of tools which would not only allow extensive exploration of all aspects of the design, but also provide building blocks that encourage the close interaction of software and hardware engineers. This approach had the advantage that valuable information was constantly being communicated between hardware and software groups during the development process.

The powerful tools which were set in place included a digital logic simulator and Computer Hardware

---

* Now with the Superconducting Super Collider

**This work performed under the auspices of the United States Department of Energy.

Description Language (CHDL), a high-speed graphics package and a knowledge-based expert inference system, all running on a powerful work station. Figure 2 shows how their integration was viewed from the developer and the user. Although all of these tools are very useful when used in isolation, their combined effect is even more powerful and versatile. For example, in order to configure, download, monitor and diagnose the "model" of the data acquisition system, a user interface was developed which accommodated these functions in a very friendly way. The requirements of this interface were in many cases identical to those of downloading, monitoring, and diagnosing the data acquisition system of an actual physics experiment. If the model is an accurate representation of the actual system, then everything that a user would like to do to the system, he would also like to do to the model. Therefore, as the model was developed, the actual software used to help run the experiment was also developed in parallel in an integrated fashion, thereby providing a universal interface accomodating the model and the "real" system.

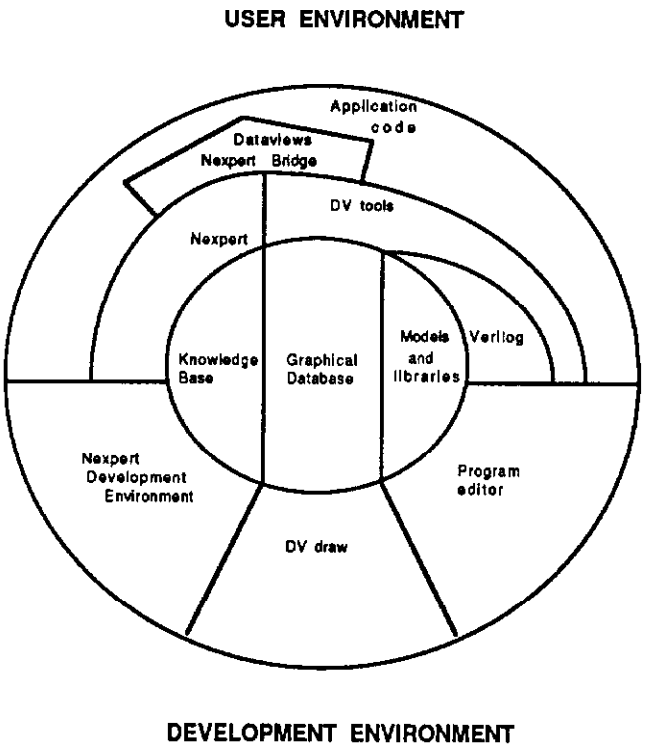


Figure 2 Tools from User/Developer Perspective

Another example of integrated systems engineering is the development of system diagnostics and their integration into the hardware design during the simulation process. Good systems diagnostics are crucial for minimizing downtime in a running experiment. In order to diagnose something, it helps to understand it. Before any hardware was actually built, diagnostic strategies were being designed and tested.

2.0 Tools

2.1 Verilog

The first tool we chose was Veriolog -XL [3], which is a digital logic simulator based on the computer hardware description language Verilog -HDL. It provides advanced simulation capabilities designed to handle complex electronic designs. It has many features which can be summarized as follows:-

(i) Different levels of abstraction; system/architectural, behavioral/algorithmic register transfer, gate/circuit
(ii) Mixed level modeling
(iii) Stochastic analysis
(iv) Interactive debugging environment
(v) Open Design Environment; library support, programming language interface (C, etc.)

These features make Verilog very useful for modeling data acquisition systems, even those which include multiple ASIC's and complex VLSI devices. Verilog has already been used successfully in the HEP community [4]. Verilog provides three kinds of graphical display; firstly a logic analyzer type display which is very useful for low level debugging, secondly a register display where the programmer can display the value of any variable or register in his HDL code, and thirdly a bar graph display for showing such things as queue occupancy and overflows in the stochastic/queue management type modeling.

2.2 DataViews

The next tool we chose was Dataviews [5], which is a powerful graphical design environment for developing custom color displays for real-time monitoring and control. This is a very important aspect of the Scalable Parallel Open Architecture Data Acquisition System; that the "User Interface" be state-of-the art. Dataviews is written in C and runs on most 32-bit workstations. It comprises two main components; a powerful drawing editor called DVdraw, and a comprehensive set of utilities called DVtools. DVdraw enables users to create and modify color pictures, and in our case these were system type diagrams of our user interface as wells as menu driven displays for user interaction.

DVtools allows the user to specify the dynamic interactions of all components on each screen and between screens, as well as how to integrate the displays into user application programs. We have also used Dataviews to "front-end" Verilog, so that the infrequent user of Verilog can come along and set up parameters for a simulation run in a user friendly way, without having to know Verilog HDL code.

2.3 Nexpert

The third tool we chose was Nexpert [6] which is a powerful "knowledge representation and reasoning" system. It includes a rule and "object-oriented" expert system shell

and was particulary attractive to us for three reasons; firstly it meets some goals we have in terms of diagnosing data acquisition systems, secondly it includes a unified database bridge which interfaces to a variety of database packages, and thirdly it has a software bridge to Dataviews. This last feature was very attractive to us since the bi-directional relationship between Nexpert and Dataviews means that by clicking on buttons on a Dataviews "view", rules fire which can cause for example, other programs to be invoked (such as reading status registers in the DAQ system), and even other "views" to be displayed (such as a lower level in the DAQ system).

3.0 Software Development

With the tools in place, there were five main areas of software development; simulation, diagnostics, embedded code,user interface and remote procedure calls.

The software effort was completely done on a SUN OS 4.0.3 based platform. The main programming language was C with a small amount of sparc assembler code. The following tools provided by SUN were also used; compiler, linker, "vi" text editor, dbx source level debugger,and the rpc generator utility(rpcgen). The amount of code generated was approximately 50000 lines and is broken down into the following main areas:

a. user interface, DVTOOLS programs, not views (20000 lines)
b. simulation code - various models (10000)
c. embedded code, for all modules (4000 lines)
d. rpc ,server and client calls (12000 lines)
e. diagnostic software - consisted of some code already accounted for,i.e. status,rpc. a general number for the decision trees was (2000 lines)
(this was only implemented to a very small degree).
f. code to generate look up tables etc. (2000 lines)

The SUN OS has the facility to generate the remote procedure calls easily through the use of the rpcgen utility. This facility allowed us to write the user programs that were to be activated over the network without having to worry about the network protocol or operations.

3.1 Modeling & Simulation

The purpose of writing models and simulating the switch-based Scalable Parallel Open Architecture Data Acquisition System was to provide a learning vehicle whereby the system designers could experiment with different architectures and control mechanisms to enable them to better understand DAQ design. An improved understanding simplifies decisions such as which operation mode provides for highest throughput, what extra electronics and software should be implemented to more efficiently diagnose failures and fix problems, etc. Modeling and system simulations assist system designers in determining throughput for different configurations, identifying potential bottlenecks, interfacing to "physics data" simulations, identifying busiest channels, selecting proper buffer sizes, determining the number of processors and processing power required, determining data rates, and many other decisions which are normally made using analytical calculations or intuition.

At the outset of the simulation experiments the goals of the exercise were specified clearly:-

a) to develop simulation models of the following functional sub-units of the switch-based DAQ system:-

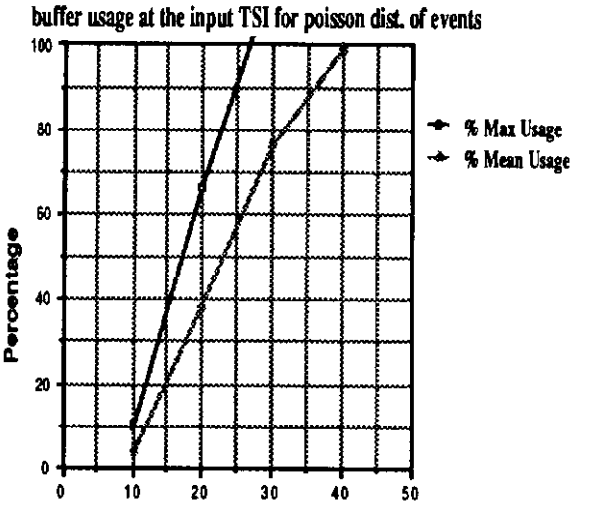
(i) trigger system interface
(ii) test transmitter module
(iii) switch
(iv) switch control
(v) input time slot interchanger
(vi) output time slot interchanger

b) to develop a "system" model of an 8 by 8 switch architecture, using each of the above functional sub-components

c) to observe the behavior of the "system" model as well as each of the functional sub-units when varying certain subsets of the following parameters:-

(i) trigger rate
(ii) event distribution over switch output channels
(iii) event size
(iv) buffer depth in the input TSI's
(v) buffer depth in the output TSI's
(vi) switch packet size
(vii) test transmitter to input TSI data transmission rate
(viii) input TSI to switch data transmission rate
(ix) switch to output TSI data transmission rate

d) to perform analyses and produce meaningful results in terms of summary plots and printouts



This graph shows the mean and maximum amount of buffer usage for different subevent sizes

The results of the simulation experiments have been reported elsewhere [7], and so we shall not include them

here. However, one interesting plot worth including here is one that shows the degradation of the switch when events are distributed unevenly over output channels instead of on a round robin basis. The plot shows how much more the buffers overflow in the TSI's when events are distributed unevenly.

3.2 Diagnostics

In the context of diagnosing a switch-based data acquisition system, we desire to narrow the problem down to a particular functional sub-unit as quickly as possible. To accomplish this goal it was important that there be some form of error detection facility built into the design, however minimal. The switch project allowed for error detection by the use of a set of diagnostic buffers which were periodically checked by the system monitor to see if any errors had been encountered.

Once an error was detected a fact gathering process took place, followed by a series of questions leading to a conclusion about the possible origin of the error. To enable an organized method of data gathering, question asking and eventually decision making we implemented a ruled based knowledge system using Nexpert. This system allowed a series of decision trees (generated from the knowledge elicited from the hardware designers during the knowledge acquisition process) to be implemented in software so that the system could pinpoint the most probable origination point of a particular error .

Because of the bridge between Nexpert and Dataviews, integration of the diagnostic facility into the existing user interface system was easy and quick. Questions are asked via the RPC mechanism and rudimentary deductions are made. For example, to determine if a problem originates before the switch or after the switch, the following sequence of steps would take place:

1. An error was detected by the embedded code resident on one of the modules. The embedded code was responsible for updating status registers that were located in local dual ported memory.
2. The system monitor, through the local monitors, are periodically polling each module and reading the respective status registers. The error is detected by the system monitor which notifies the operator of the error condition.
3. The operator receives a error message through the user interface, and then activates the diagnostic system,which is fully integrated into the user interface.
4. The diagnostic system, which has the error, starts working its way thru the decision trees that were earlier defined. The decision trees offer a path through the different modules.The different modules are polled, only as needed, and the diagnostic system starts narrowing the list of possible originators of the error to a small subset.
5. The operator is then notified of the most logical place to start troubleshooting. The expert system is able to diagnose down to the board level.

If all nodes are "seeing" the problem, then the problem is probably pre-switch or on the input to the switch itself, whereas if only one channel is experiencing a problem then the problem is post-switch or on the outputs of the switch itself. Either way this simple rule has performed the first "binary chop" on our total DAQ system. At present, this is the least developed part of the project, but still remains a positive area for investigation and development.

Another possible path to error correction is the use of the diagnostic buffers that were included at the design phase of the modules. These diagnostic buffers could be used to look at the history of events that have been passed through each module. The system included a way of selectively dumping, into the diagnostic buffers, only header information or all information including data.

3.3 Embedded Software

The embedded software was meant to be fast and efficient. There was no need for a real-time operating system to be installed on the modules, mainly because there was to be only one task running. The embedded code for all modules included a set of common functions:

a. boot program
b. downloading
c. message protocol over vme
d. status,for diagnotic purposes

The same executable code for these common functions was downloaded to each the same type modules thereby reducing unnecessary duplicate development. Very little information was downloaded for configuration purposes simply because much of the information needed to transfer data through the system is carried along with the data.

3.3.1 Embedded code on specific boards

Test Transmitter boards: main purpose was to simulate a front end sub-detector system. The embedded code on this board included functions to generate a set of data that was to flow through the system, with the basic header information that the system needed to route the event fragments to the proper destination level 3 farm processors.

Trigger System Interface (TRIF) board: act as the trigger supervisor combined with the trigger system. The embedded code on this board would generate the trigger and notify the test transmitters as well as the input TSI's at predetermined time intervals. This triggering rate could be changed dynamically via the message passing protocol from the system monitor.

Input TSI boards: this module received the event fragments from the Test Transmitters and fed them into the switch on a packet by packet basis. The embedded code on these boards was responsible for checking headers, receiving trigger information and routing the data into the correct buffers in preparation for "switching".

Output TSI boards: this module received the event fragments from the switch and fed them into the appropriate buffer in preparation for transmission to the level 3 farm as a complete event. The embedded code on these boards was responsible for sending the data, buffer by buffer, to the level 3 farm ("event-building"). In our system the output

TSI also simulated the level three processor farm, which required the maintenance of the event request link.

3.4 User Interface

The graphical user interface was developed to allow a non-textual representation of the system,which we felt was more intuitively understood. The interface was developed using the graphic package DATAVIEWS. This package was C based and consisted of two major areas of effort.

The look of the view itself was done using DVDRAW. This allowed the developer to actually see the way the view would look ,attach variables to the view and allow the developer to see the layout of the view easily.

The view was controlled by a C program that was responsible for drawing,updating and destroying each view.This programming control is the DVTOOLS portion of Dataviews. It allows a connection between the view and the remote procedure call system,which collected information from the real DAQ system or the simulation system.

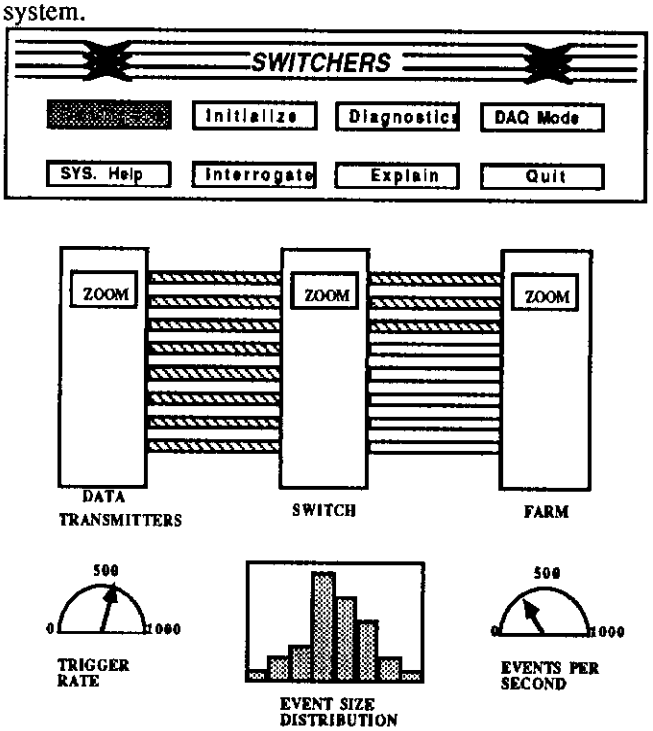


Figure 3 Example of Dataviews graphics

Through use of a mouse the user could jump thru the levels of view and quickly get an image of the system,while the system was running. The user was also able to change parameters, reset the system, and examine status statistics.

The graphical user interface was hierarchical in design and implementation. The top view allows the user to select such operations as simulations or system software, or to reconfigure the system. Through use of the mouse or keyboard the user could change parameters and send the information to the system from the selected view.

The development of the User Graphical Interface was started well before the design on many of the modules was complete. This was possible because the system could be designed,tested, and debugged using the simulation model as a "true" system.

3.5 Remote Procedure Calls

The remote server and client programs used to write and read from the SWITCH DAQ system were compiled and linked with the skeleton XDR routines produced from a program definition file that was compiled with rpcgen. The client and server stub routines interface with the RPC library and effectively hide the network from the calling programs. Several remote procedure calls were written for initializing the SWITCH DAQ system by downloading the embedded code, lookup tables and initialization of local variables. Other programs make client calls to read from the various parts of the system for data display and diagnostics.

Conclusions

The prototype switch-based DAQ system has met the performance objectives, both in terms of hardware and software. In terms of integrated systems engineering, this project has demonstrated the importance of having powerful tools and integrating them properly in order to ensure wise design decisions are made at all phases of the development. There is still a long way to go, but the goal of having reliable DAQ systems in HEP experiments of the future looks reachable.

Acknowledgements

The authors wish to gratefully acknowledge Mark Bowden, Ed Barsotti and all members of the design team of the Scalable Parallel Open Architecture Data Acquisition System.

References

1. "A High-Throughput Data Acquisition Architecture Based on Serial Interconnects", M. Bowden, et al, IEEE Transactions on Nuclear Science, Vol. 36, No. 1, February 1989, p760-764.
2. "Verilog-XL",Cadence, Lowell,Massachusetts.
3. "Simulation of a Macro-pipelined Multi-cpu Event Processor for Use in Fastbus", M.F.Letheren, A. Marchioro, F Slorath, CERN, Geneva, Switzerland.
5. "Dataviews", V.I. CORP,Amherst,Massachusetts.
6. "Nexpert",Neuron Data Inc.,Palo Alto, California.
7. Simulating and Modeling of Data Acquisition Systems for Future HEP Experiments, A.W.Booth, M.Bowden, E.J. Bartsotti, D.Walsh & D. Black, IEEE Transactions on Nuclear Science.